Learning of geometric phase field representations

Yannick Kees

Geboren am 14.05.1998 in Daun

Oktober 2022

Masterarbeit Mathematik

Betreuer:

Zweitgutachter: -

Mathematisches Institut

Mathematisch-Naturwissenschaftliche Fak der

Rheinischen Friedrich-Wilhelms-Univer Bonn

Danksagung

Mit dieser Seite möchte ich mich bei allen Personen bedanken, die auf unterschiedliche Art und Weise zum Gelingen dieser Arbeit beigetragen haben.

Contents

Vorwort

1 Surface reconstruction problem

1.1 Storing data

storing geometirc data, objects,4 explicit [Pou20]

- **Point clouds.** Point clouds are unordered set of points, where every point consists of different features, like its coordinates, the color at this point or the corresponding normal vector at this position. The big advantage of point clouds is, that this form of representation is closest to the raw data output that we would gain from sensors, like depth cameras or laser scanners. In these cases, the points are sampled from the surface of a geometrical object. However, we run into problems if we want to visualize them. Rendering the bare point cloud is very easy, but often we are much more interested in reconstructing the original object. Since point clouds do not contain much information of the geometry or topology of the underlying objects, it they do not produce water tight surfaces right away. So for this, we might need to convert the point cloud into one of the other representation formats.

- **Voxeling.** If we want to represent data in the 2D-case, we often use images as representation. Images can be seen matrices, where every entry describes a same sized square, called pixel. Voxeling is the approach of doing this one dimension higher. Therefore, we consider now a three dimensional matrix, where every entry describes the properties of a certain cube. The big advantage of this representation is that it gives us a vector space structure, which allows us to perform mathematical operations on the data, like addition or convolution. However, a downside to this is, that we need to specify the resolution, so the size of the individual cubes. If we choose the size too small, due to the cubical growing of numbers of cubes, we will have to expect very high memory requirement, even if we are only dealing with spatial data. On the other hand, if we choose the cube size too big, we get too pixelated results, which can lead to a loss of information but also does not look very natural.

- **Meshing.** Another way of producing surfaces is by using meshes. We start by sampling a point cloud from the underlying object and then we connect some of the nearby points with edges. We do this until we have approximated the object with simple small geometric figures. One example for 3D-data is by using triangles. Finding a valid triangulation of points is quiet challenging. One of the most popular and most used methods for this problem is called Delaunay Triangulation ([Del34]), but this problem is still not fully solved. The difference to the voxeling approach is that meshes can only be used to represent the surfaces of objects but are not suitable for solid objects. But since the sizes of the different polygons in the mesh do not need to be the same, we get a better approximation, yielding in more natural looking visualization.

- **Image view.** The last approach can only be used for 3D data. The idea is that we take images from the object from many angles and so reducing the 3D data problem by one dimension to a 2D problem.

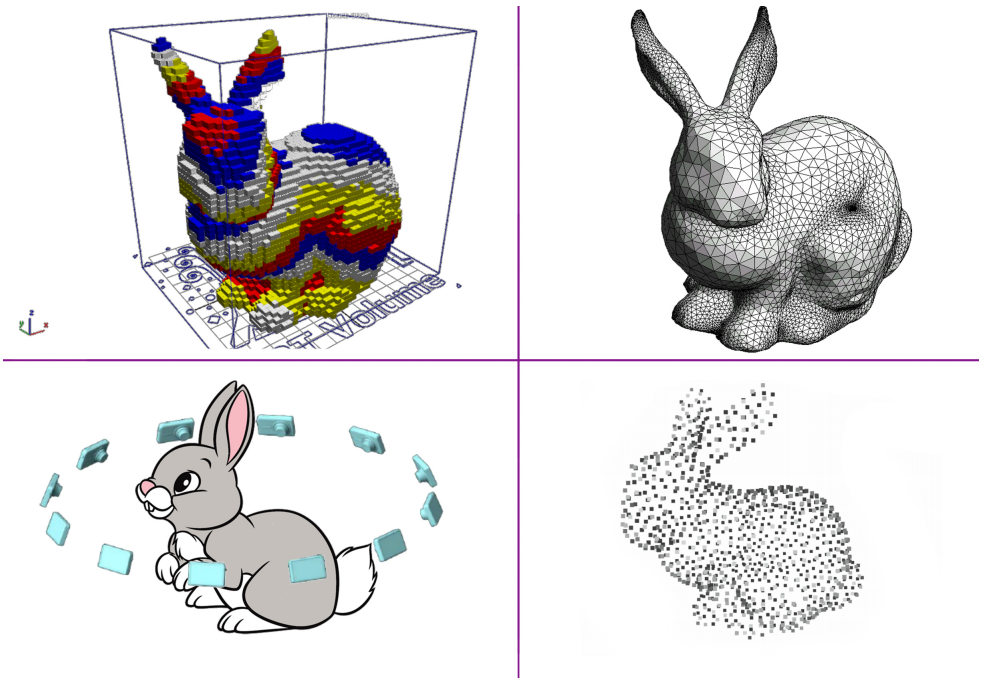

Figure 1: The four explicit ways of storing data

Instead of these representations, we will focus on a less visual representation for data, the **implicit representation**. For this, we will consider the given data as the level quantity of a function.

1.2 Initial problem

Suppose we have given a space $\Omega \subseteq \mathbb{R}^d$ and a surface $\mathcal{S} \subseteq \Omega$ and from this surface we sample a point cloud $X = \{p_1, \dots, p_n\}$. For now we will only consider the coordinates as features of the point cloud, i.e. X is a subset of Ω. Now our goal will be, given this point cloud, to reconstruct the original surface $\mathcal{S}$. In order to do this, we have to think about an easy way, to describe the set $\mathcal{S}$ properly. While finding a direct representation through some parametrisation is rather difficult, the easier way to use an implicit representation. In our case, we want to write the set as a zero-level set. That means, we want to find a function $u : \Omega \to [-1, 1]$, such that it holds

$$\mathcal{S} = \left\{ x \in \Omega \;\middle|\; u(x) = 0 \right\}.$$

Note, that this representation also gives us a disjoint division of our space Ω into three parts: our surface $\mathcal{S}$, the interior of the surface $\mathcal{I} = \left\{ x \in \Omega \;\middle|\; u(x) < 0 \right\}$ and the exterior $\mathcal{O} = \left\{ x \in \Omega \;\middle|\; u(x) > 0 \right\}$. If u is a solution to the surface reconstruction problem, then we can easily see that $-u$ is also one. So the choice of interior and exterior is really arbitrary.

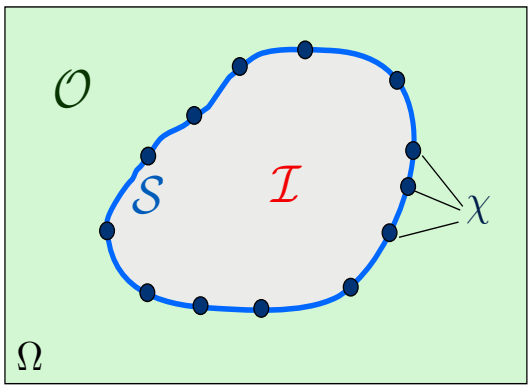

Figure 2: A 2D-example for the setting of surface reconstruction

For the function u, that we have used for the implicit representation, we only care about its zero-crossings and not so much about what happens outside of $\mathcal{S}$. That means for the outside behavior we obtain an additional degree of freedom. Based on that, we can consider three major classes of functions, that we could use for the implicit representation.

- **Indicator functions.** The easiest way of describing S is by the indicator function of its compliment $u_{IF} = \mathbb{1}_{S^c}$. This function is 0 if and only if a point lies on the surface and is 1 else wise. If we use this for the surface reconstruction problem, the interior will always be the empty set. That is why we will neglect this approach for now.

- **Occupancy functions.** These are all the functions that can be written as

$$u_{OF}(x) = \begin{cases} 1 & x \text{ is inside} \\ 0 & x \text{ is on the boundary} \\ -1 & x \text{ is outside} \end{cases}.$$

  The image of these functions is also discrete and they can be seen as a more general version of indicator functions.

- **Signed distance functions (SDF).** These functions also encode the distance between the given point and the original surface $\mathcal{S}$. If we have given an occupancy function u_{OF} we get a signed distance function u_{SDF} by

$$u_{SDF}(x) = u_{OF}(x) \cdot d(x, \mathcal{S})$$

  where $d(x, S)$ is the distance from the point x to S. So a signed distance functions maps every point to its positive distance to S, if the point is inside and to the negative distance if it is outside. They can be seen as a continuous version of the occupancy functions.

Note that given a signed distance function it is very easy to compute an occupancy function, but the other way round is challenging, if we do not know S in advance.

1.3 Classical approach to surface reconstruction problem

1.4 Deep neuronal networks

Before we explain, how we want to learn occupancy functions or signed distance functions, we want to start with a broad overview on artificial intelligence based on the first chapter in [Agg18]. In general **Artificial Intelligence** is a subsection of computer science and mathematics. It deals with machines mimicking functions typically associated with human cognition. We consider it as the attempt of a computer to recreate certain decision-making structures of the human brain.
The techniques and processes that are used for solving these problems are summarized under the term **machine learning**. Instead of solving a problem explicit, we put all the data in a generic algorithm, that then can solve the problem. So machine learning analyses data to „learn“and then use this knowledge to make decisions. We differ between three categories, based on how the machine learning algorithm performs the learning.

(i) **Supervised Learning.** We supply the algorithm with inputs and the desired outputs. The output variables are often called labels or categories. The algorithm should approximate a mapping functions, that predicts the category for a given observation. It is common for classification models to predict a continuous value as the probability of a given example belonging to each output class The probabilities can be interpreted as the likelihood or confidence of a given example belonging to each class. A predicted probability can be converted into a category by selecting the class label, that has the highest probability.
For example an email of text can be classified to be either „spam“or „no spam“. So we train the algorithm by feeding it with E-Mails from which we already now weather they are spam or not and in the end the program should get a new mail and make a decision on its on.

(ii) **Unsupervised Learning.** We only supply inputs and no outputs. The algorithm adjust itself in such a way, that similar inputs cause similar outputs. It identifies the patterns and differences in the input without any external assistance. It is used for clustering problems or detecting anomalies.

(iii) **Reinforcement Learning.** Constant feedback is given into the model to adapt to the environment. There is a performance evaluation at each step to improve the model.

Since the late 2000s **deep learning** has been the most successful approach to supervised learning problems. It uses **artificial neuronal networks** to solve the problems we stated before. Any neuronal network always consists out of same three parts: the inputs form an input layer, the middle layers which perform the processing of the inputs are called hidden layers and the output forms the output layer. The hidden layers consist of **neurons**, that perform mathematical operation on their associated inputs. Each input to a neuron is associated with a different weight, which affects the computation. So neuronal networks represent functions of their given input by propagating the computed values from one layer to the next until they reach the final output layer. The key to neuronal networks is to adjust the weights in such a way, that the function represented by it solves the initial problem. In general we can think of artificial neuronal networks universal function approximaters.
We now consider the easiest case of an ANN, a **multi-layer perceptron (MLP)**. MLPs are the most widely used architecture for neuronal networks. To understand how they work we take a look at

what happens in the neurons of the MLP. The neurons do two important things. First they calculate a weighted sum of their incoming data by performing an affine linear transformation to it. Afterwards the transformation is inserted into an **activation function** σ. A common choice for an activation function is the **Rectified Linear Unit (ReLU)** $\sigma(x) = \max\{0, x\}$. This function is the identity for all positive inputs and maps all negative values to 0. So if we consider a single neuron, that gets n inputs x_i and is associated with weights w_i, the output of this neuron would be

$$\sigma\left(\sum_{i=1}^{n} w_i x_i + w_{n+1}\right).$$

For evaluating a MLP, we repeat this calculation for every neuron until we reach the output layer.

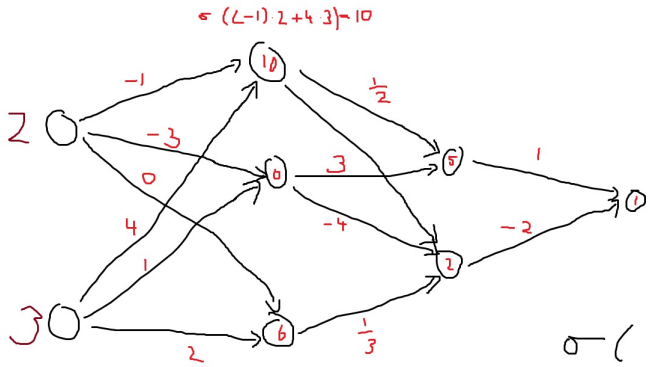

Figure 3: Example of the forward propagation of a neuronal network with two hidden layers, ReLU activation function and no bias

Now we want to use the MLP to solve supervised learning problems. Let f the by function representing the architecture of the network, $\vec{w}$ be the weights of the network summarized as a vector, $(\vec{x}_i)_{i=1}^n$ be the training data and $(\vec{y}_i)_{i=1}^n$ be the corresponding training data. For the training of the MLP, we need to measure how good the prediction are, the network is currently making. This means we need a way to measure the overall error of it. Now we can choose a **loss functional** $\mathcal{L}$, that is a map, that takes the predicted labels and the actual labels as input and returns the total error. The choice of the loss functional is crucial in what exactly the network should learn. Then we can write the supervised learning problem as the following minimization problem.

$$\min_{\vec{w}} \mathcal{L}\big((f(\vec{x}_i, \vec{w}))_{i=1}^n, (\vec{y}_i)_{i=1}^n \big)$$

For solving minimization problems there are plenty of numerical algorithms, that can be used, like gradient descent. That means we have reformulated the supervised learning problem into a problem, that we can handle and therfore found a solution to the original problem.
We now want to specify more, what a solution actually is. By increasing the number of layer and neurons, the network is able to learn more complex functions on the training set. That means if we have a network that is too small, we will not be able to get good results. However, making the network to big, will provide good performance on the training data, but does not guarantee good performance on unseen test data. This problem is called **overfitting** and arises if the network learns a function,

that is too specifically matched to the underlying training data.

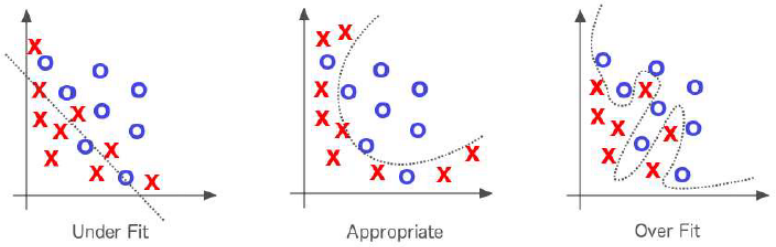

Figure 4: 2D example of under/over-fitting

One way of reducing the overfitting error is by adding a regularization term to the loss functional

$$\mathcal{L} = \mathcal{L}_{data} + \mathcal{L}_{reg}.$$

Here $\mathcal{L}_{data}$ is the loss between the predicted labels and the actual labels, and $\mathcal{L}_{reg}$ is function measuring the complexity of the network. So a appropriate solution function should map the inputs close to their labels, but should also be as simple as possible.

1.5 Learning implicit surface representations

Now we want to get back to the surface reconstruction problem. The idea is to consider it as a supervised learning problem, where the training data are the points of the point cloud, since we know, how our function should behave there. That means the training data are the points $x_i \in X$ and the corresponding labels are $y_i = 0$.
As the neuronal network f for the implicit representation of $\mathcal{S}$ we choose a MLP. By the previous chapter, the only thing we need to specify now is how to choose the loss functional. That means we can reformulate the surface reconstruction problem as:

Find a loss functional $\mathcal{L}$ such that $S = \{x \in \Omega \mid f(x, w_*) = 0\}$ with $w_* = \arg\min_w \mathcal{L}\big((f(x,w))_{x\in X}, X\big)$.

Next we want to get rid of the weights in the problem formulation. Therefore we need the following result for multi-layer perceptrons.

Theorem 1.1 (Universal approximation theorem):
Let $\Omega \subset \mathbb{R}^d$ be compact. For any map $f : \Omega \to \mathbb{R}$ continuous, there is a sequence of functions $(f_n)_{n\in\mathbb{N}}$ representing a MLP with ReLU activation functions and $d+2$ hidden layers, such that

$$\lim_{n\to\infty} \|f - f_n\|_{C_0(\Omega)} = 0$$

You can find a proof of this theorem in [?]. The big takeaway from this is, that we can approximate any function with a neuronal network. For our loss function, this mean we no longer need to minimize over some weights, but rather consider a minimization problem over the whole space of functions. That means we solve:

Find a loss functional $\mathcal{L}$ such that $S = \{x \in \Omega \mid f_*(x) = 0\}$ with $f_* = \arg\min_f \mathcal{L}\big(f, X\big)$.

Since we will need large networks for this, we need to address the overfitting problem. So let $\mathcal{L} = \mathcal{L}_{data} + \mathcal{L}_{reg}$ as before. The data loss is responsible for making the function 0 along $\mathcal{S}$. The regularization

error is responsible for making the function as smooth as possible elsewhere.
In this thesis we are going to discuss several losses

1.6 Learning shape spaces

2 Mathematical prerequisites

2.1 Gamma convergence

In the context of machine learning our goal is always to find a minimizer for a given functional. These minimizers may be very hard to compute, so instead we want to find an approximation of this functional, that converges against it. The problem with that is, that our usual definition of convergence does not provide us any information on how the corresponding minimizers behave. Ideally we would want that minimizers of the approximation to converge to a minimizer of the original functional. For that to work, we need a new definition on what convergence of functionals should actually mean.

Definition 2.1:
Given a metric space (X, d) and a family of functionals $F_n : X \to \overline{\mathbb{R}}$, we say F_n **$\Gamma-$converges** to some $F : X \to \overline{\mathbb{R}}$, if for every $x \in X$ the following two properties hold:

(i) For every sequence $(x_n)_{n\in\mathbb{N}} \subset X$ with $x_n \longrightarrow x$ it holds

$$F(x) \leq \liminf_{n\to\infty} F_n(x_n) \tag{1}$$

(ii) There exists one sequence $(\tilde{x}_n)_{n\in\mathbb{N}} \subset X$ with $\tilde{x}_n \longrightarrow x$ and

$$\limsup_{n\to\infty} F_n(\tilde{x}_n) \leq F(x)$$

The first property in the Γ-convergence definition is often called the lim inf-property and the second one lim sup-property. The sequence we used in the lim sup property is called recovery sequence. With this new definition we can now investigate how minimizers behave under this convergence.

Lemma 2.2:
Let $F_n : X \to \overline{\mathbb{R}}$ be a family of functions and let x_n be a minimizer of F_n. Suppose that $(x_n)_{n\in\mathbb{N}}$ converges to x and F_n Γ-converges to F. Then x is a minimizer of F.

Proof: Let $y \in X$ arbitrary. By the second property of Γ-convergence we can find a sequence (y_n) with $y_n \longrightarrow y$ and $\limsup_{n\to\infty} F_n(y_n) \leq F(y)$. We obtain

$$F(x) \overset{(1)}{\leq} \liminf_{n\to\infty} F_n(x_n) \leq \limsup_{n\to\infty} F_n(x_n) \leq \limsup_{n\to\infty} F_n(y_n) \leq F(y)$$

Since y was arbitrary, we can take the minimum and get $F(x) \leq \min_{y\in X} F(y)$.

□

So if we have a sequence of functionals $\mathcal{F}_\varepsilon$ with corresponding minimizers u_ε and we want to find a minimizer for $\varepsilon \to 0$ we need two things: First the $\mathcal{F}_\varepsilon$ should Γ-converge and second the u_ε need to converge in the usual sense. Showing that the sequence of minimizers does converge can be really challenging, since we do not actually want to compute them. That is why, we want to find some additional properties of the functionals, under which we can drop this condition.

Definition 2.3:
A sequence of functionals $F_\varepsilon : X \to \overline{\mathbb{R}}$ is called **equi-coercive** if for every $t \in \mathbb{R}$ we can find a compact set K_t such that

$$\{F_\varepsilon \mid F_\varepsilon \leq t\}.$$

2.2 Bounded variation and surface perimeter

[Giu84]

Definition 2.4:
Let $\Omega \subset \mathbb{R}^d$ be open and $f \in L^1(\Omega)$. We define the **total variation** of f as

$$\int_\Omega |Df| := \sup_{\substack{g \in C_0^1(\Omega, \mathbb{R}^d) \\ \|g\|_\infty \leq 1}} \int_\Omega f(x) \mathrm{div}(g)(x)\, \mathrm{d}x$$

If $\int_\Omega |Df| < \infty$ we say f has bounded variation and we denote the space of all functions with bounded variation as $BV(\Omega)$.

Note that $\int_\Omega |Df|$ is just some notation and does not actually need that f is differentiable, but can be applied to a much bigger group of functions. Yet, this notion we use suggest that the variation can be seen as some form of integration over the derivative. In fact, the next lemma will show that if f is continuous differentiable, then this is actually holds.

Lemma 2.5:
If $f \in C^1(\Omega)$ then the total variation is the integral over the absolute value of the gradient

$$\int_\Omega |Df| = \int_\Omega |\nabla f(x)|\, \mathrm{d}x.$$

Proof: In order to prove this, we first apply integration by parts to obtain the following expression.

$$\int_\Omega |Df| := \sup_{\substack{g \in C_0^1(\Omega, \mathbb{R}^d) \\ \|g\|_\infty \leq 1}} \int_\Omega f(x) \mathrm{div}(g)(x)\, \mathrm{d}x = - \sup_{\substack{g \in C_0^1(\Omega, \mathbb{R}^d) \\ \|g\|_\infty \leq 1}} \int_\Omega \sum_{i=1}^d \frac{\partial f}{\partial x_i} g(x)_i\, \mathrm{d}x$$

Since g is bounded by 1, we get

$$\int_\Omega |Df| \leq \int_\Omega |\nabla f(x)|\, \mathrm{d}x.$$

For the other inequality we choose as test function $g = \frac{\nabla f}{|\nabla f|}$. This is bounded by 1 and we obtain

$$\int_\Omega |Df| \geq \int_\Omega |\nabla f(x)|\, \mathrm{d}x,$$

which yields the assertion.

□

If we go back to the surface recontruction problem, our goal was to find a functional, that would minimize the boundary of some domain. If a boundary is smooth enough, we already know that we can measure its size with the correct dimensional Hausdorff measure. If not, the concept of total variation allows us now to define the size of them, by looking at the total variation of the corresponding indicator function.

Definition 2.6:
Let $I \subset \Omega$. Then we define the **surface perimeter** by

$$\mathrm{per}_\Omega(I) := \int_\Omega |D\mathbb{1}_I|$$

The following lemma show, why we can really see the surface perimeter as a more general version of the measure of the boundary.

Lemma 2.7:
Suppose $I \subset \Omega$ has a C^2-boundary, then we can write the perimeter as

$$\mathrm{per}_I(\Omega) = \mathcal{H}^{d-1}(\partial I).$$

Proof: Let $g \in C_0^1(\Omega)$ with $\|g\|_\infty \leq 1$. Then we can apply the Gauss-theorem and with $n : \partial I \to \mathbb{R}$ being the function, that maps a vector to its outgoing normal, we obtain

$$\mathrm{per}_I(\Omega) = \int_\Omega \mathbb{1}_I \mathrm{div}(g)(x)\,\mathrm{d}x = \int_I \mathrm{div}(g)(x)\,\mathrm{d}x = \int_{\partial I} \langle g(x), n(x)\rangle\,\mathrm{d}x$$

And now with the real version of Cauchy-Schwarz we get

$$\int_\Omega \mathbb{1}_I \mathrm{div}(g)(x)\,\mathrm{d}x \leq \int_{\partial I} \underbrace{|g(x)| \cdot |n(x)|}_{=1\cdot 1}\,\mathrm{d}\mathcal{H}^{d-1}(x) = \mathcal{H}^{d-1}(\partial I)$$

For the other direction let $N : C^1(\mathbb{R}^n, \mathbb{R}^n)$ be an extension of the outer normal n. Then we consider all functions g that are of the form $g = \phi N$ for a test function $\phi \in C_0^\infty(\Omega)$ with $\|\phi\|_\infty \leq 1$. Then

$$\mathrm{per}_I(\Omega) = \sup_{\substack{g \in C_0^1(\Omega, \mathbb{R}^d) \\ \|g\|_\infty \leq 1}} \int_{\partial I} \langle g(x), n(x)\rangle\,\mathrm{d}\mathcal{H}^{d-1}(x) \geq \sup_{\substack{\phi \in C_0^\infty(\Omega, \mathbb{R}^d) \\ \|\phi\|_\infty \leq 1}} \int_{\partial I} \phi(x) \underbrace{\langle N(x), n(x)\rangle}_{=1}\,\mathrm{d}\mathcal{H}^{d-1}(x) = \mathcal{H}^{d-1}(\partial I)$$

□

2.3 Euler-Lagrange formula

3 Finding Energys

3.1 Phase transition problem

Before we continue on the surface reconstruction problem, we want to consider a very related problem from physics: the phase transition problem. In this setting, we have a space Ω and in this we put two different liquids. We want to find out, how these liquids behave, therefore we denote the density of the two liquids by a function $u : \Omega \to [-1, 1]$. For some position $x \in \Omega$ the case $u(x) = 1$ means that in this position we have the first liquid and $u(x) = -1$ means we have only the second liquid. The case $u(x) = 0$ means we have a perfect mixing ratio of the two liquids in this point. What we will see in reality is that the both fluids will arrange themselves in such a way, that the intersection among them has minimal area. That means, we want that almost every point within Ω should be either 1 or -1. To formulate this more mathematically, we need the following definition.

Definition 3.1:
A **double-well potential** W is a continuous function $W : \mathbb{R} \to [0, \infty)$, such that

(i) $W(x) = 0$ if and only if $x \in \{1, -1\}$.

(ii) There exist $L, R > 0$ such that $W(z) > L|z|$ for every $|z| \geq R$.

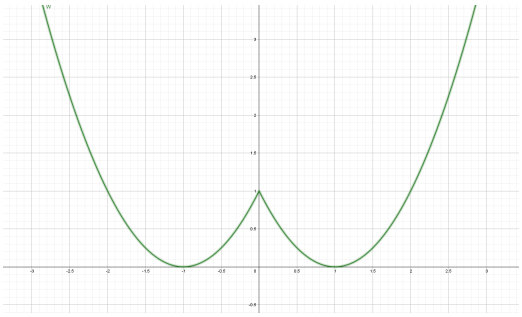

Figure 5: Graph of a double-well potential

Let W be a double-well potential. With the reasoning from before, we want $W(u)$ to be zero as much as possible. More mathematically speaking, we are looking for a solution for the following minimization problem:

Phase transition

$$\min_{u \in L^1(\Omega,[-1,1])} \int_\Omega W(u(x))\,\mathrm{d}m(x)$$

$$\text{s.t.:} \quad \int_\Omega u(x)\,\mathrm{d}m(x) = m$$

The secondary condition here comes from the fact, that we input a certain fixed amount of the two liquids, so m is a constant value.
There is one problem with the minimization problem stated above and that is, it does not minimize

the area of interference of the liquids. In fact, every decomposition of the space Ω, where u is 1 on the one part and -1 in the other part, is a minimum for this functional. So we want to change the minimizer in such a way, that there is really only one minimum. We do so by adding a regularization term to it.

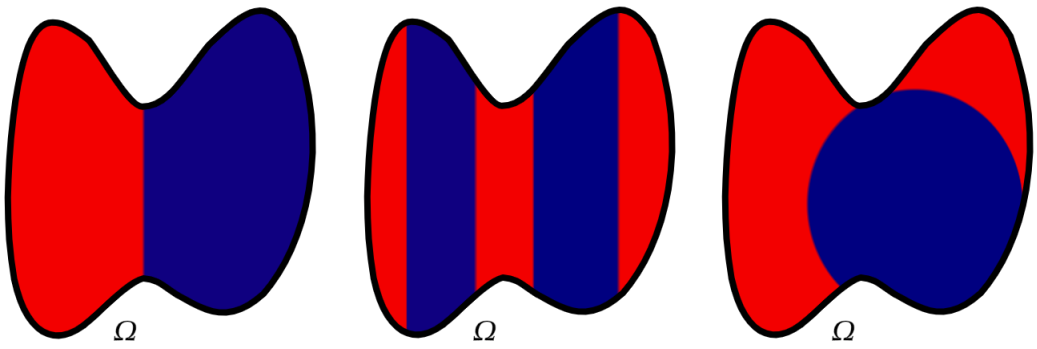

Figure 6: All these images are local minima of the phase transition energy, but we want the left one to be the only minimum

Adding a regularization term here eventually does the same as adding a regularizer term in the context of machine learning. In both cases we have given a whole set of minima and want to reduce this to just one minimum, which should be the most „simple“one. Now in our case we decide to use a **Dirichlet regularizer**, which is just the integral over the absolute value of the gradient. This means we end up with the following minimization problem, that was found by Modica/Mortola.

Modica-Mortola

$$\min_{u\in L^1(\Omega,[-1,1])} \mathcal{M}_\varepsilon(u) := \int_\Omega \frac{1}{\varepsilon} W(u(x)) + \varepsilon \|\nabla u(x)\|^2 \,\mathrm{d}m(x)$$

$$\text{s.t.:} \qquad \int_\Omega u(x)\,\mathrm{d}m(x) = m$$

We now actually get a whole family of minimization problems, that depend on a single parameter $\varepsilon > 0$. We can see, that as ε decreases, the contribution of the double well potential increases, while the one of the regularizer term decreases. It can be shown, that any of the functionals $\mathcal{M}_\varepsilon$ has a unique minimizer and that this sequence of minimizers converges to limit, we can directly compute.

Theorem 3.2 (Modica-Mortola):
Under some assumptions on Ω, there exists a functional $\mathcal{M}: \ L^1(\Omega,[-1,1]) \to \mathbb{R}\cup\{\infty\}$, such that $\mathcal{M}_\varepsilon$ is Γ-converging to $\mathcal{M}$, where

$$\mathcal{M}(u) := \begin{cases} c|Du|(\Omega) & u \in BV(\Omega,\{1,-1\}) \\ \infty & \text{else} \end{cases}$$

3.2 Modica Mortola for Surface reconstruction

Now we want to use the approach we derived for the phase transition problem onto the surface reconstruction problem. We already know, that there are multiple ways to address this problem. There has been a lot of research done, trying to create loss functionals for either an occupancy function or a signed distance function. But for now we want to use the approach by Yaron Lipman. In this paper he proposed the idea of learning first an occupancy function and from this create a signed distance function. So the final loss he uses to create a solution, manages to combine features of both of these losses. But before we do this, we want to state, what a solution in this case actually is. Ideally the minimizer u_0 of the loss functional should fulfill the following properties.

(i) **Proper occupancy.** We want to solve the surface reconstruction problem by using an occupancy function, so the image of u_0 should be 1 or -1 almost everywhere. To say this differently we want $u_0 \in BV(\Omega, \{\pm 1\})$.

(ii) **Zero reconstruction loss.** The original point cloud should be in the set of zero crossings of u_0, that is $u_0(x) = 0$ for all $x \in X$.

(iii) **Minimal perimeter.** If we would just demand the first two properties to hold, every arbitrary decomposition of Ω into an exterior and interior would be a solution. From all these decompositions, we want that functions, that has a minimal perimeter. So u_* should minimise $\mathrm{per}_\Omega(\mathcal{I})$.

(iv) **Easy way of getting a SDF.** Since the representation of a occupancy function can be challenging, we also want a way, that given the occupancy function, produces a signed distance function.

If we apply the approach of the surface reconstruction problem, we can regard the first liquid as the interior of the surface and the second liquid as the exterior. Using then the Modica-Mortola functional would already ensure the proper occupancy and minimal perimeter property. Since we do not have a mass constraint we replace it by the zero reconstruction loss, by summing over the absolute values averaging integral over small balls of the original point in the point cloud. We also do not treat this as a secondary condition, but as a summand in the functional.

PHASE$_0$

$$\min_{u \in H^1(\Omega,[-1,1])} \mathcal{F}_\varepsilon(u) := \int_\Omega \frac{1}{\varepsilon} W(u(\xi)) + \varepsilon \|\nabla u(\xi)\|^2 \, \mathrm{d}m(\xi) + \frac{\lambda}{|X|} \sum_{x \in X} \left| \fint_{B_\delta(x)} u(\xi) \, \mathrm{d}m(\xi) \right|$$

Here λ is some positive constant, depending only on ε, that we specify later. By minimizing the functional, we want to enforce the averaging integral to be 0 around these small balls of radius δ. Note, that this is a stronger condition, that just wanting the function to be zero in the underlying points from the point cloud, because the integral becomes only zero, if some parts of it cancel out. This should force the function to change its sign along the surface S.

Figure 7: Minimizing the integral over some ball around a point $x \in X$ yields in a change of sign of the function u.

Theorem 3.3 (Lipman):
If we choose $\lambda..$, then $\mathcal{F}_\varepsilon \xrightarrow{\Gamma} \mathcal{F}_0$, where

$$\mathcal{F}_0(u) = \begin{cases} C\mathrm{per}_\Omega(\mathcal{I}) & u \in BV(\Omega, \{-1,1\}) \text{ and } \mathcal{Z}(u) = 0 \\ \infty & \text{else} \end{cases}$$

Proof:

□

3.3 SDFs and the Eikonal equation

In the previous chapter we have successfully found a way of solving the surface reconstruction problem by learning an implicit representation of the underlying surface through an occupancy function. But in the beginning we said that we would actually prefer the representation through a signed distance function. So now we want to find a way of using the functionals $\mathcal{F}_\varepsilon$ and the corresponding minimizer from before to compute a signed distance function.
Therefore we have to adjust the one left degree of freedom that plays into the construction of $\mathcal{F}_\varepsilon$, and that is the double well potential. For the previous theorems to work, any suitable double well potential would work, but for the next computations, we need to explicitly state a function. So from now on let

$$W(x) := x^2 - |2x| + 1$$

We can see that this is actually a double well potential. Let u_ε be a minimizer of $\mathcal{F}_\varepsilon$, that we obtain using exactly this double-well potential. We already know that these converge to a suitable occupancy function, so now we want to apply a transformation map to these minimizer and see what happens. Therefore we choose the so called logarithm transform.

Definition 3.4:
Let $\varepsilon > 0$. For the minimiser u_ε of $\mathcal{F}_\varepsilon$ we define its **log-transform** by

$$w_\varepsilon(x) = \begin{cases} -\sqrt{\varepsilon}\log(1 - u_\varepsilon(x)) & u_\varepsilon(x) \geq 0 \\ \sqrt{\varepsilon}\log(1 + u_\varepsilon(x)) & u_\varepsilon(x) < 0 \end{cases}$$

We see that the log-transform is continuous and does not change the sign of the original function. The goal is to show, that w_ε will now give us an approximation of a SDF. To proof this, we will not use

the variational problem $\min \mathcal{F}_\varepsilon$ directly, but rather show it through the partial differential equation, that characterizes their minima.

Lemma 3.5:
Let $Q \subseteq \Omega - \bigcup_{p\in\mathcal{P}} B_\delta(p)$, with $u_\varepsilon(x) \neq 0$ for all $x \in Q$. Then on Q the function u_ε is smooth and fulfills the following partial differential equation

$$-\varepsilon\Delta u_\varepsilon + u_\varepsilon - \operatorname{sign}(u_\varepsilon) = 0.$$

Proof:

□

Theorem 3.6:
Let Q be the domain as in Lemma (3.6), then on Q the log-transform w_ε satisfies

$$-\sqrt{\varepsilon}\Delta w_\varepsilon + \operatorname{sign}(u_\varepsilon)\left(\|\nabla w_\varepsilon\|^2 - 1\right) = 0$$

Proof:

□

3.4 Other losses for OF's and SDF's

3.5 Image reconstruction problem

3.6 Ambrosio Tortorelli theorem

4 Numerical implementations

4.1 First results

4.2 Improving the network

Index

List of Figures

References

[Agg18] Aggarwal, Charu C.: *Neural Networks and Deep Learning.* Springer International Publishing, 2018 https://www.ebook.de/de/product/33161337/charu_c_aggarwal_neural_networks_and_deep_learning.html. – ISBN 3319944622

[Del34] Delaunay, Boris N.: Sur la sphère vide. In: *Bulletin of Academy of Sciences of the USSR. 7* (1934), Nr. 6, S. 793–800

[Giu84] Giusti, Enrico: *Minimal surfaces and functions of bounded variation.* Boston : Birkhäuser, 1984. – ISBN 3764331534

[Pou20] Poux, Florent: *How to represent 3D data?* https://towardsdatascience.com/how-to-represent-3d-data-66a0f6376afb. Version: Mai 2020. – Last Access: 24.11.2021